CISM exercise II: run diagnostic test cases

From Interactive System for Ice sheet Simulation

Contents

Introduction

In this exercise, we will run a few of CISM's higher-order test cases, which span a wide range of flow regimes. Since the test cases are small and can be run on a single processor, we will run them "interactively" (that is, without submitting the jobs to the queue). First, make a directory on the scratch space to run the code and store the output, for example,

```
mkdir /scratch2/username/cism
```

(where "username" is your username). Now copy the relevant directories there. From within the "CISM-LANL-4-2011/tests/" subdirectory,

```
cp -r higher-order /scratch2/username/cism/
```

This subdirectory contains various test cases for the newer, "higher-order" dynamical core in CISM. Within the "higher-order" directory you will see the following subdirectories,

```
dome/                # parabolic shaped dome with simple boundary conditions
ismip-hom/           # ISMIP-HOM test suite
ross/                # Ross ice shelf test case
shelf/               # ice shelf test cases on simplified domains
```

along with a few other files. You may want to look over the tests/higher-order/README file at some point but most of the necessary information from that file is contained on this page. While you are welcome to explore any of the test cases on your own (most of them can be run by simply following the instructions in the README files within each subdirectory), for this exercise we will pick a few representative examples that can be run relatively quickly.

Before we can run the code we first need to start an interactive session using one node,

```
msub -I -A s11_cesm
```

Once your prompt has been returned to you, make sure that you are still in the directory you want to be in on your scratch space. Since we've started up a new session you will need to re-source the environment script,

```
source /usr/projects/cesm/cism/cism-env-csh
```

CISM uses Python to read and write input/output netCDF files. Here, we need a fairly specific set of Python toolboxes. To make sure that you have access to the correct version of Python, type "python". You should see the following first few lines:

```
Enthought Python Distribution -- www.enthought.com
Version: 7.0-2 (64-bit)
Python 2.7.1 |EPD 7.0-2 (64-bit)| (r271:86832, Nov 29 2010, 13:51:37)
```

If instead you see,

```
Python 2.4.3 (#1, Sep  8 2010, 11:37:47)
[GCC 4.1.2 20080704 (Red Hat 4.1.2-48)] on linux2
```

let us know so that we can alter a few things and give you access to the correct version of Python. To escape out of Python, use <CTRL><D>.

The Dome test case

This is a very simple test case, simulating the three-dimensional flow field within an isothermal, 3d, parabolic shaped dome with no-slip basal boundary conditions and zero flux lateral boundary conditions. To run the test case, first change into the "dome" subdirectory,

```
cd /scratch2/username/cism/higher-order/dome/
```

Now make a virtual link to the executable file you built in the first exercise. All of the test cases use the **simple_glide** executable, which is built from the *simple_glide.F90* driver in *example-drivers/simple_glide/src/*. To link to it from your scratch space type,

```
ln -s /path/to/your/project/space/CISM-LANL-4-2011/example-drivers/simple_glide/src/simple_glide ./
```

You can then execute the test with

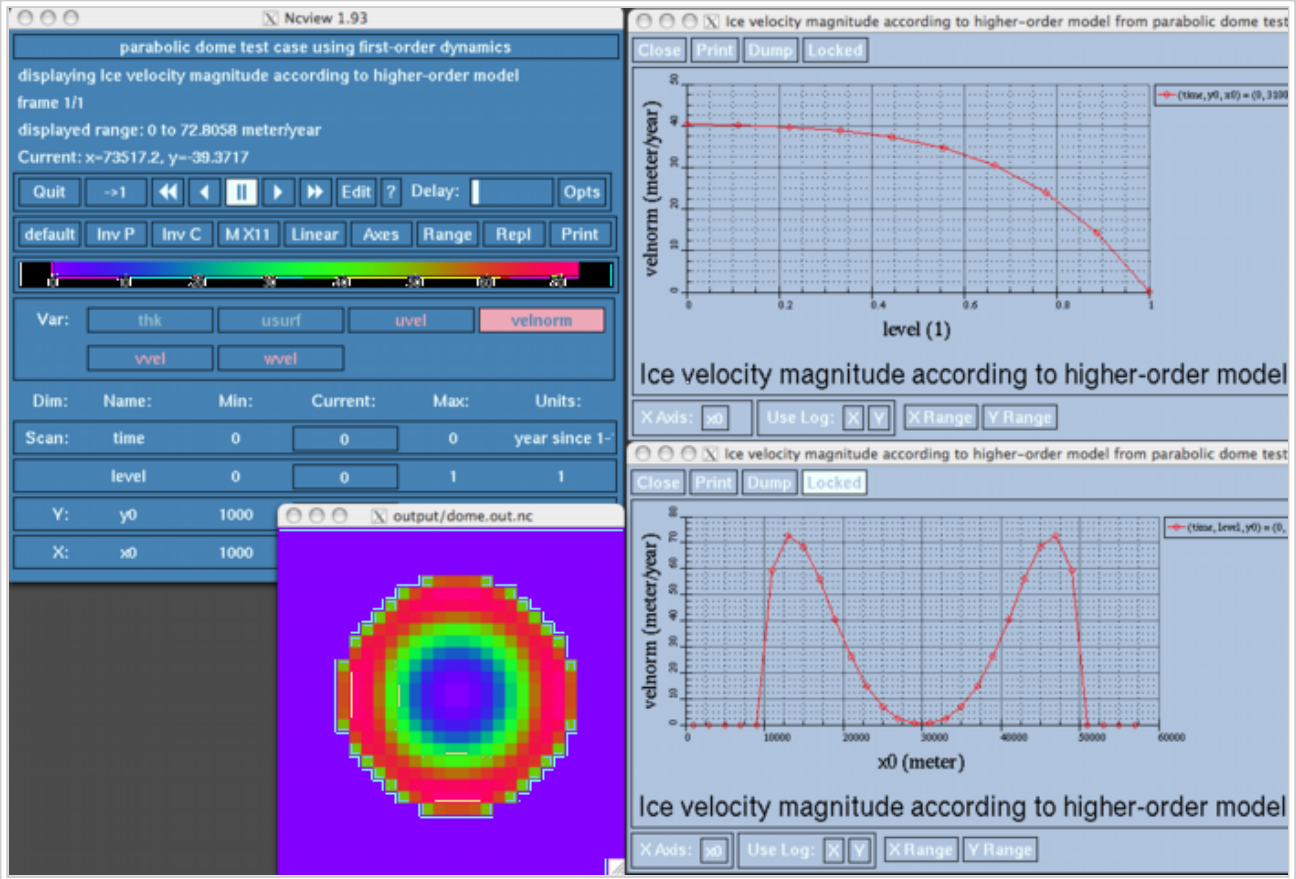
```
python dome.py
```

The call to the python script first builds an input netCDF file in the *output/* subdirectory and executes **simple_glide**. In general, whenever **simple_glide** is executed, it expects to be followed by the name of a text file with the ".config" extension. If such a file is not specified, you will usually see something like

```
Enter name of GLIDE configuration file to be read
```

Here, the "dome.config" script from within this directory is passed to **simple_glide** by the "dome.py" python script.

While there are numerous default settings in the code, in general it will need a configuration file of some sort to specify various things like grid size and spacing, various solver options, boundary conditions, etc. A good way to get a feel for what these options are and what parts of the code they trigger is to look in the ".config" file for an option you want to understand (e.g. "evolution = 3") and then "grep" for that option in the file *glide_types.F90* in the *libglide/* subdirectory from within the main directory where you built the code.



Dome test case: Screen grab of model output from dome test case using NCVIEW. Shown are the velocity magnitude in map view (color plot), the surface speed across the ice dome, and a vertical velocity profile from approximately half way between the ice divide (center) and the margin (click for higher-resolution image).

As the code runs, you will see some output to the screen like

```
(dH/dt using incremental remapping)
time =    0.0000000

Running Payne/Price higher-order dynamics solver

iter #     resid (L2 norm)       target resid

1          223.257             0.100000E-03
2          223.150             0.100000E-03
3          216.909             0.100000E-03
4          203.843             0.100000E-03
5          180.486             0.100000E-03
6          149.276             0.100000E-03
7          116.333             0.100000E-03
...
39         0.341224E-03         0.100000E-03
40         0.226718E-03         0.100000E-03
41         0.150639E-03         0.100000E-03
42         0.100090E-03         0.100000E-03
43         0.665039E-04         0.100000E-03
```

The output you see here is fairly standard. It tells us the following information

1. Which solver we are using to evolve the ice thickness (if at all). Here, we see that we are using *incremental remapping* (which we will discuss further later on). However, looking in the ".config" file we see that the start and end times are identical, so no geometric evolution will take place; we are simply after a diagnostic solution here.
2. The current time step we are solving for.
3. The dynamics scheme we are using (here, the Blatter-Pattyn equations as formulated and solved by Payne and Price).
4. the non-linear iteration number, the current residual (the L2 norm of the vector **r = Ax - b**), and the target residual

Note that when the residual is less than or equal to the target residual, the nonlinear iterations are halted and we have a "converged" solution (i.e. we have the answer). Here it took 43 nonlinear iterations to arrive at a converged solution.

You can look at the model output using any convenient netCDF file viewer. A python-based netCDF file viewer, *viewNetCDF.py* is included in top level of the *tests/higher-order* subdirectory. Another common viewer installed on many machines (including the machine used here) is *NCVIEW*. To examine the output file using NCVIEW, type

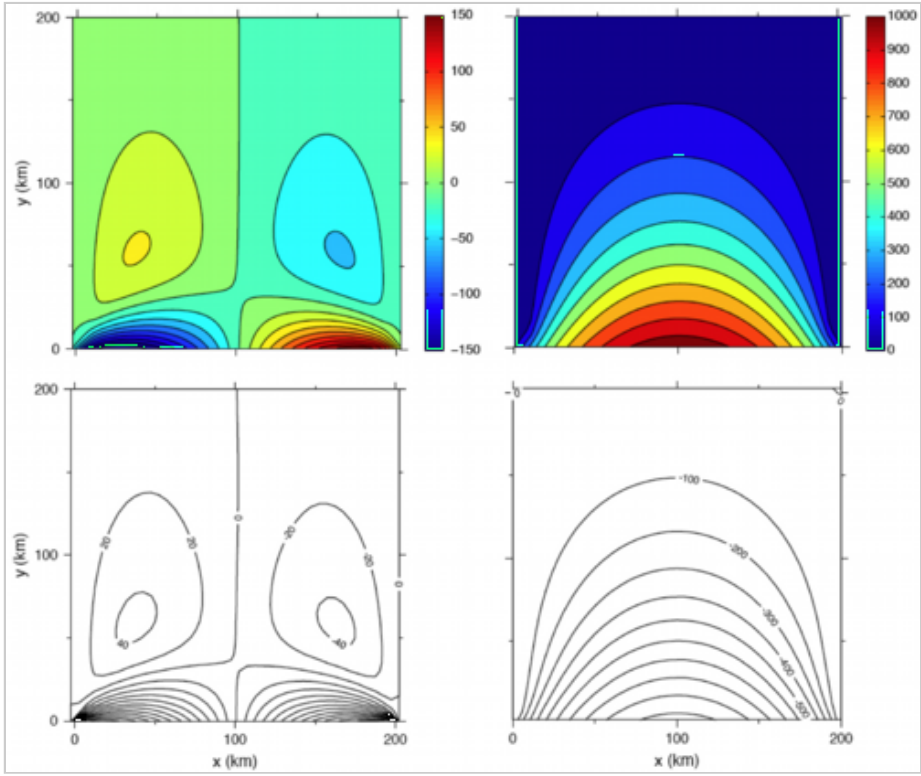
```
ncview output/dome.out.nc
```

Your output for the variable *velnorm* (the ice speed) at level "0" (sigma coordinate level 0, which is the upper surface of the ice sheet), should look something like what is shown in the figure labeled **Dome test case**. Take a minute to play around with the different buttons on NCVIEW to see what they do. You can step through the vertical levels of any of the 3d model output fields, click on the color contour plots to obtain 2d profiles, change the color scheme, and for variables that change in time, run simple "movies" showing a variables evolution over time (we will do this later).

Admittedly, this is not a very exciting test case. However, as a sanity check it should confirm whether or not the model is working as expected and shows that, for a very "shallow-ice" like test case, the model indeed reproduces something that looks very much like shallow-ice flow.

The Confined Shelf test case

Now we will go all the way to the other end of the spectrum and demonstrate that the exact same model can also accurately reproduce ice shelf flow. In the following idealized ice shelf test case, there has been no change at all in the governing equations of the model. The only thing that has changed is the geometry and the boundary conditions of the test problem. Instead of a parabolic dome with no basal slip we now have a flat, floating slab of ice with free-basal slip, zero-flux boundary conditions on three sides, and open-ocean (ice shelf) boundary conditions on the fourth side.



Confined shelf test case: Along (right) and across (left) flow velocities for the confined shelf test case. The upper panel (color) shows results using CISM and the lower panel shows results from the EISMINT ice shelf intercomparison project (http://homepages.vub.ac.be/~phuybrec/eismint/iceshelf.html) for the same test case. Solid black contour lines are the same in both plots (click for higher-resolution image).

To run the test case, change from the "higher-order/dome/" subdirectory into the "/higher-order/shelf/" subdirectory. There are two idealized ice shelf tests cases here, *confined-shelf.py* and *circular-shelf.py*. To run the confined shelf test case, proceed with a similar set of steps as when running the dome test case. First, link to the **simple_glide** executable as you did before, then run the test with,

```
python confined-shelf.py
```

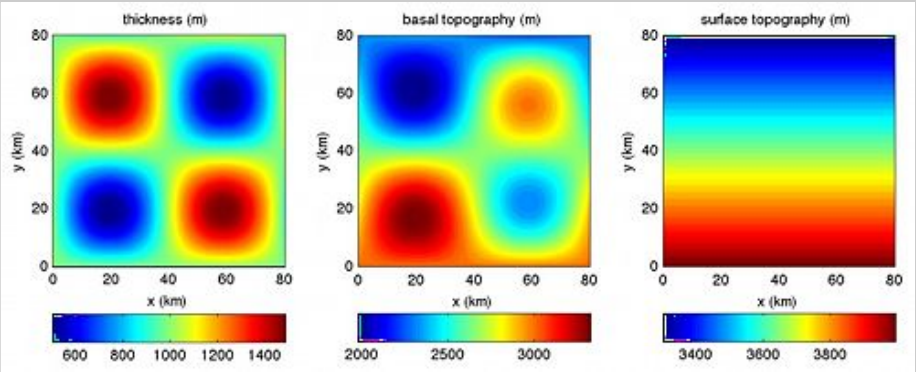
As in the previous test case, you should see gradually decreasing residuals as screen output. When the test completes, examine the output with

```
ncview output/confined-shelf.out.nc
```

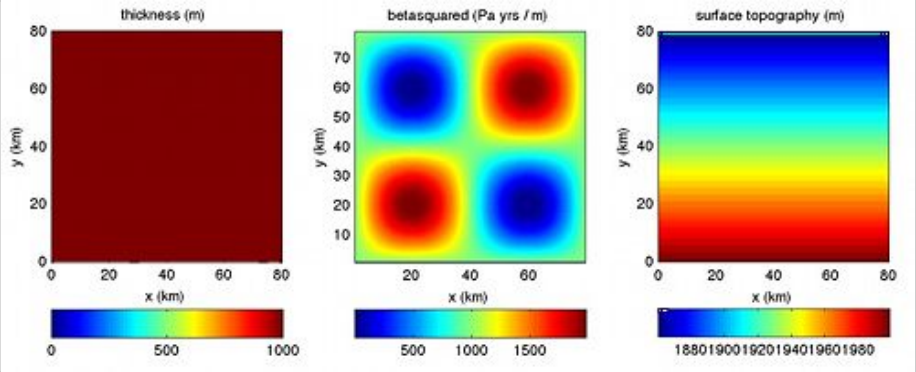
A color contour plot of CISM output (made in Matlab) is shown in the figure labeled **Confined shelf test case**. The black and white contour plot shows output for the same experiment using an SSA (ice shelf) model. It is from experiment 3 (page 7) of the EISMINT (http://homepages.vub.ac.be/~phuybrec/eismint.html) (European Ice Sheet Model InTercomparison) ice shelf intercomparison project (http://homepages.vub.ac.be/~phuybrec/eismint/iceshelf.html) documentation (http://homepages.vub.ac.be/~phuybrec/eismint/shelf-descr.pdf) . We will return to this experiment and add some additional complexity to it in a later exercise.

If there is time, you may also want to try running the "circular-shelf" experiment, which demonstrates that CISM can also implement an accurate ice shelf boundary condition for an ice shelf front with a non-trivial shape in map view (i.e. one for which the shelf-front normal vectors are not parallel to coordinate directions).

The ISMIP-HOM test cases



ISMIP-HOM A Setup: Doubly periodic basal roughness with no sliding. Ice thickness, basal topography, and surface elevation are shown. At the lateral boundaries, velocities are doubly periodic (click for higher-resolution image).



ISMIP-HOM C Setup: Doubly periodic basal traction coefficient with sliding. Ice thickness, basal topography, and surface elevation are shown. At the lateral boundaries, velocities are doubly periodic (click for higher-resolution image).

The last set of diagnostic problems we will look at are from the ISMIP-HOM (http://homepages.ulb.ac.be/~fpattyn/ismip/) test suite, which was specifically designed for "higher-order" models and nicely demonstrates the difference between higher-order and 0-order (or "shallow ice") models. While the test suite includes a total of 6 tests we will look only at the tests for diagnostic solutions on idealized, three-dimensional domains. Each of these tests (A and C) includes a subset of 6 tests for a range of domain lengths.

Both tests consist of a uniformly sloping slab of ice with periodic lateral velocities in the *x* and *y* directions (i.e. in map plane). For test A, the basal topography varies periodically in *x* and *y* directions and the there is a no-slip basal boundary condition. For test C, basal sliding is allowed. The basal traction coefficient varies periodically in *x* and *y* and the thickness is uniform throughout the domain. While the *amplitude* of the variations (topography in A and traction coefficient in C) is the same for all tests, the *wavelength*, λ, is decreased by a factor of two for each successive test. For λ=160 km, the velocity solutions are essentially equal to those from a 0-order shallow ice model. When halving λ to 80 km, then to 40, 20, 10, and finally 5 km, the higher-order components of the stress balance become successively more important to the velocity solution. Figures 1 and 2 below show relevant input data for each of the two experiments for λ = 80km. Here, in the interest of time, we will only run tests for the first three wavelengths in the series (160, 80, and 40 km).

Running the model test cases

To run the experiments, we will use some python scripts developed by colleagues at the University of Montana (also, see this link). As with the other test cases, several Python scripts set up the necessary netCDF input and output files. The python scripts simplify things here by allowing you to run and plot the results from multiple tests and multiple domain wavelengths sequentially. In addition, they plot CISM output relative to the model means and standard deviations from the actual benchmark study of Pattyn et. al (2008) (http://www.the-cryosphere.net/2/95/2008/tc-2-95-2008.html) . This is a great convenience (as anyone who has ever done this on their own will

attest to!). First, move into the *tests/higher-order/ismip-hom* subdirectory. To execute test A, for λ=160, 80, and 40 km, type

```
python runISMIPHOM.py --exp=a --size=160,80,40
```

As with the other test cases above, you should see some screen output showing model residuals decreasing as the nonlinear iterations proceed (***Did you remember to make a virtual link to the **simple_glide** executable?***?).

Plotting model output

To compare CISM output from your model runs with that from the ISMIP-HOM benchmark study of Pattyn et. al (2008) (http://www.the-cryosphere.net/2/95/2008/tc-2-95-2008.html) , we will execute the python plotting scripts in a similar manner. For test A, type

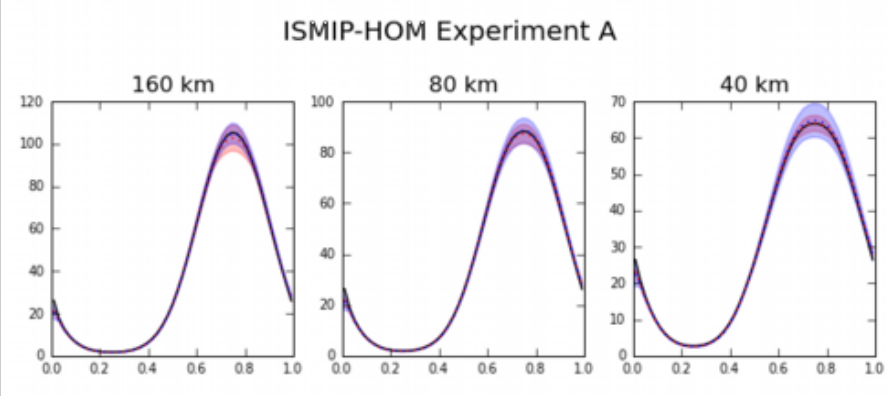
```
python plotISMIPHOM.py --exp=a --size=160,80,40
```

Your output figure will have a ".png" extension and will be placed in the *output/* subdirectory. It should look something like the figure here labeled **ISMIP-HOM A Output**. There is a simple image viewer on the cluster where we have been running the code (the classic XV). To use it to view the output form you test case type,

```
xv output/ISMIP-HOM-A-glm1.png
```

Now go through the same set of steps for test case C (again, with wavelengths of 160, 80, and 40 km). You should get a figure that looks like the figure labeled **ISMIP-HOM-C Output**.

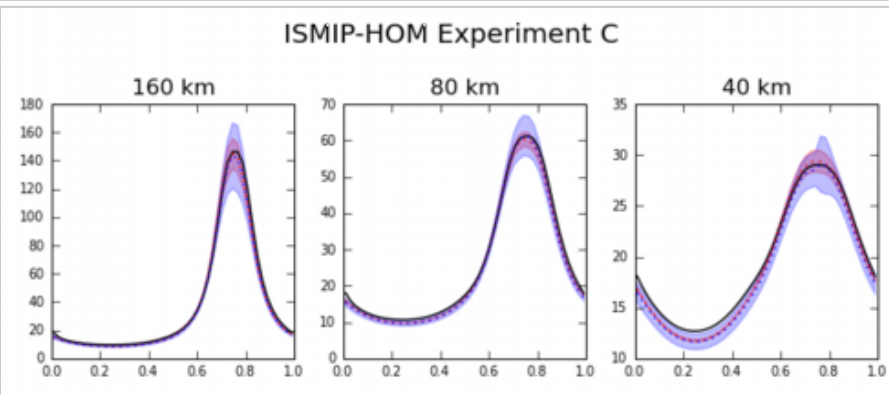
For additional information on running and plotting results for the ISMIP-HOM test suite, see the README file in the *tests/higher-order/ismip-hom* subdirectory.



ISMIP-HOM A Output: CISM output for experiment A run for wavelengths of 160, 80, and 40 km. Black solid lines are CISM output, blue (red) dashed and shaded areas show the mean and standard deviation from all first-order (Stokes) models participating in the benchmark exercise. Model velocity is shown on the vertical axis and normalized distance along the domain, at y=0.25*λ, is shown on the horizontal axis (click for higher-resolution image).

Additional Exercises

ISMIP-HOM A with shallow-ice dynamics



ISMIP-HOM C Output: CISM output for experiment C run for wavelengths of 160, 80, and 40 km. Black solid lines are CISM output, blue (red) dashed and shaded areas show the mean and standard deviation from all first-order (Stokes) models participating in the benchmark exercise. Model velocity is shown on the vertical axis and normalized distance along the domain, at y=0.25*λ, is shown on the horizontal axis (click for higher-resolution image).

To clarify the importance of the higher-order stresses in the model velocity solutions, it is instructive to go back and re-run one of the above tests using the shallow-ice model. To do this, we first need to edit some of the configuration file options in the file *ishom.a.config*. Copy the original file to a backup version first (e.g. cp ishom.a.config ishom.a.config.orig). Now open *ishom.a.config* with your favorite editor (e.g. VI or Emacs) and look for the following sections:

```
[options]
flow_law = 2            # constant and uniform rate factor
periodic_ew = 1         # doubly periodic lateral boundary conditions
periodic_ns = 1
evolution = 3
```

```
[ho_options]
diagnostic_scheme = 1   # Payne/Price 1st-order dynamics
which_ho_babc = 4       # no-slip basal boundary conditions
which_ho_efvs = 0       # nonlinear eff. visc. w/ n=3
which_ho_sparse = 1     # use SLAP GMRES for linear solver
```

To implement 0-order shallow ice dynamics rather than first-order dynamics, change the following flags in the

options and *ho_options* sections,

```
[options]
evolution = 0           # now SIA dynamics!
```

```
[ho_options]
diagnostic_scheme = 0   # now SIA dynamics!
```

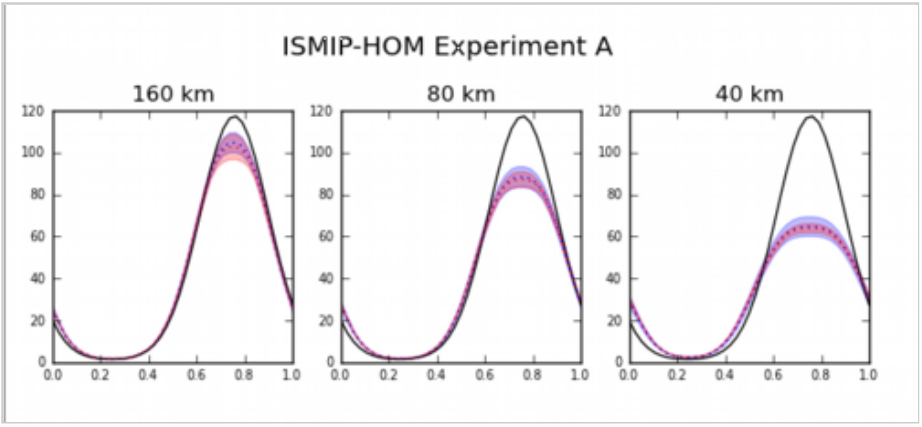
Now re-run the ISMIP-HOM test case

```
python runISMIPHOM.py --exp=a, --size=160,80,40
```

You won't see any output, but you will probably notice that the model gets through both of these tests much more quickly than when using the first-order stress balance (one good thing about shallow ice dynamics, they are computationally cheap!). When the model is done running, plot the results again,

```
python plotISMIPHOM.py --exp=a, --size=160,80,40
```

Your results should look something like what is shown in the figure labeled **ISMIP-HOM-A SIA Output**. Can you explain why the velocity field from the shallow ice model is identical despite the change in the wavelength of the basal topography for the three experiments? As far as the shallow-ice model is concerned, these three domains are all identical because the flow rate is controlled only by the local slope and ice thickness (which is the same despite the different wavelengths of the basal topography).



ISMIP-HOM A SIA Output: CISM output for experiment A, run for wavelengths of 160, 80, and 40 km **using shallow-ice dyanmics**. Black solid lines are CISM output, blue (red) dashed and shaded areas show the mean and standard deviation from all first-order (Stokes) models participating in the benchmark exercise. Model velocity is shown on the vertical axis and normalized distance along the domain, at y=0.25*λ, is shown on the horizontal axis (click for higher-resolution image).

ISMIP-HOM A: Newton versus Picard

The increasing difficulty of the ISMIP-HOM experiments as the domain wavelength decreases provides a good opportunity to demonstrate the differences between handling the model nonlinear with a Picard versus a Newton iteration (as discussed in more detail on the model solution page). Because the current Newton solver in CISM is still under development, we have to make a few more simplifications here to compare the two. In particular, we have not yet implemented periodic boundary conditions in the Newton iteration in which case we will have to turn these "off" for the Picard iteration as well. As above, we need to edit a few sections in your your **original ishom.a.config** file (which was hopefully copied before you made the previous edits). First, to run the test cases using Picard, change the periodic flags in the *options* section of your .config file as follows:

```
[options]
periodic_ew = 0         # Now zero-velocity rather than doubly periodic!
periodic_ns = 0
```

No re-run the test with the Picard iteration and the new boundary conditions. To get an approximate total time for the run and to dump the output to a .txt file (in case we want to plot it later on), use

```
nohup time python runISMIPHOM.py -e a -s 40,20,10 > picard-log.txt &
```

Also notice that we are now running a few of the shorter wavelength test cases in order to work the model a little bit harder.

Do the same but using the Newton iteration instead. For this case we need to add an additional flag to the *ho_options* section of the .config file:

```
[ho_options]
which_ho_nonlinear = 1  # add this flag to call JFNK for nonlinear iteration rather than Picard!
```

To re-run the test case, time the model run and save the output, use

```
nohup time python runISMIPHOM.py -e a -s 40,20,10 > newton-log.txt &
```

Note that the "nohup" command sets your job to running in the background so that you can do other things while you wait for it to complete. To check the status of your job, type

```
jobs -l
```

If your job is still running you will see something like

```
72195 Running     nohup time python runISMIPHOM.py -e a -s 40,20,10 > newton-log.txt &
```

where the first number is the job ID.

When your jobs have both completed, look in the *scratch/* subdirectory for your log.txt files. You should have a record of the iteration count for each job and also a record of the total time to run the job at the very bottom. You should notice that it takes ~4x fewer nonlinear iterations to reach converged solutions. The Newton iteration in this version of the code has not been optimized yet, so there is an additional "cost" associated with using it. Thus, you may notice that the overall savings in computational time is only about ~25% relative to Picard. In other developmental versions of the code for which the Newton iteration has been better optimized the computational time savings is usually a factor of ~2-5x (e.g. see the figure in **this**) section.

Go to the third set of exercises.

Return to main coarse page

Retrieved from "http://websrv.cs.umt.edu/isis/index.php/CISM_exercise_II:_run_diagnostic_test_cases"

- This page was last modified on 26 April 2011, at 22:30.